Proposta di Progetto: Compilatore per Confuc-IO

Francesco Petrillo
Matricola: NF22500044
Gennaio 2026

Introduzione

Confuc-IO è un linguaggio di programmazione sperimentale caratterizzato da una **semantica volutamente controintuitiva**, dove le keyword e gli operatori non corrispondono ai loro significati convenzionali. L'obiettivo è creare un banco di prova per testare la robustezza di modelli generativi di codice (LLM) di fronte a nomenclatura e sintassi divergenti dalla semantica attesa.

Specifiche del Linguaggio

Mapping Keyword

Keyword Convenzionale	Keyword Confuc-IO
if	func
while	Return
for	If
switch	func
return	*
bool	While

Mapping Tipi

Tipo Convenzionale	Tipo Confuc-IO
int	Float

Tipo Convenzionale	Tipo Confuc-IO
float	String
string	Float
bool	While

Mapping Operatori

Operatore Convenzionale	Operatore Confuc-IO
+	/
-	~
/	+
>	=
<	#
*	Bool
==	@@
=	@

Mapping Parentesi

Parentesi Convenzionale	Parentesi Confuc-IO
( )	{]
[ ]	(}
{ }	[ )

Commenti: Indicati con il simbolo È

Pipeline di Compilazione

Il compilatore seguirà la seguente pipeline di elaborazione:

1. **Analisi Lessicale:** Tokenizzazione del sorgente secondo le regole di Confuc-IO
2. **Analisi Sintattica:** Parsing tramite Lark per generare l'AST
3. **Analisi Semantica:** Type checking e validazione delle regole di scoping

4. **Generazione Codice:** Traduzione dell'AST in LLVM-IR tramite llvmlite
5. **Ottimizzazione:** Applicazione dei pass di ottimizzazione LLVM (su richiesta)
6. **Produzione Output:** Generazione dell'eseguibile (su richiesta)

Strumenti e Tecnologie

- **Linguaggio di implementazione:** Python 3.x
- **Parser/Lexer:** Lark (generazione automatica)
- **Code Generation:** llvmlite (per LLVM-IR)
- **Toolchain LLVM:** Per ottimizzazione e compilazione a eseguibile

---

Opzioni di Compilazione

Opzioni di Output

Il compilatore supporterà le seguenti opzioni di configurazione per controllare gli artefatti generati:

- `--output-ast` : Genera e salva l'AST del programma
- `--output-llvm` : Genera il codice LLVM-IR (file `.ll`)
- `--output-executable` : Genera il file eseguibile finale

Queste opzioni possono essere combinate per ottenere più artefatti da una singola compilazione.

Opzioni di Ottimizzazione

Il compilatore supporterà diversi livelli di ottimizzazione LLVM:

- `-O0` : Nessuna ottimizzazione (default)
- `-O1` : Ottimizzazione leggera
- `-O2` : Ottimizzazione media
- `-O3` : Ottimizzazione massima

Modello di Scoping e Funzione Main

- **Scope:** Unico scope globale con **shadowing proibito**
- **Funzione main:** Esplicita, denominata `side` , non accetta parametri e non restituisce valore
- **Built-in:** Implementazione minima di funzioni built-in per I/O (dettagli in fase di sviluppo)

Esempio di Programma Confuc-IO

```
side {
  Float x @ 5
  Float y @ 3
  Float z @ x / y
  func {z = 8] [
    y @ y ~2
  )
]
```

Spiegazione

1. Dichiara due variabili numeriche `x` e `y` rispettivamente a 5 e 3
2. Assegna a `z` il risultato di `x / y` (l'operatore `/` in Confuc-IO è l'addizione, quindi 5 + 3 = 8)
3. Utilizza una condizione `func` (equivalente a `if`) per verificare se `z > 8` (dove `=` rappresenta il confronto `>`)
4. Se la condizione è vera, modifica `y` tramite l'operatore `~`(sottrazione in Confuc-IO, quindi 3 - 2 = 1)

Fasi di Sviluppo

1. Definizione formale della grammatica BNF in Lark
2. Implementazione del lexer e parser con Lark
3. Costruzione dell'AST e del type checker
4. Generazione del codice LLVM-IR tramite llvmlite
5. Implementazione dei pass di ottimizzazione LLVM
6. Testing e validazione
7. Generazione della documentazione finale

Conclusione

Il progetto proposto costituisce un'occasione per esplorare le fasi critiche della compilazione e testare la capacità dei modelli generativi di affrontare scenari semanticamente inusuali. La scelta di Lark e llvmlite garantisce una base solida e moderna per la realizzazione. Le opzioni di compilazione e ottimizzazione forniscono flessibilità sia per lo sviluppo che per l'analisi dei risultati.